Facultad de Ciencias

DESARROLLO DE UNA APLICACIÓN EN SALESFORCE PARA LA GESTIÓN DE UN NEGOCIO
(SALESFORCE`S APP DEVELOPMENT FOR BUSINESS MANAGEMENT)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Pablo Herrero Escudero

Director: Patricia López Martínez

Septiembre – 2021

Resumen

A día de hoy, la gente que se dedica a las ventas y los negocios cuenta con múltiples opciones y soluciones tecnológicas que ayudan a la labor de su trabajo. Necesitan poder almacenar datos sobre sus clientes, proveedores, ventas etc. y necesitan hacerlo de forma fácil y eficiente.

Uno de los grandes descubrimientos tecnológicos en los últimos años ha sido el almacenamiento masivo de datos en la nube. La capacidad de tener acceso a esos datos en cualquier momento, desde diferentes dispositivos y desde cualquier lugar es clave para mejorar tanto la eficiencia como la facilidad de acceso antes mencionados.

Desde este punto de vista, este trabajo tiene el objetivo principal de desarrollar una aplicación en la nube que cubra las necesidades básicas de negocio para una empresa que tenga que interactuar constantemente con el cliente, recopilando datos e información y tratando así de agilizar procesos que tradicionalmente llevarían más tiempo.

En concreto, se desarrolla una aplicación para un bufete de abogados. La aplicación permite realizar actividades de gestión de expedientes, clientes y contactos y cuenta con una agenda personal para cada usuario de la aplicación con calendario, eventos e informes.

La aplicación se ha desarrollado sobre la plataforma Salesforce y se han utilizado las siguientes tecnologías: Aura Components, marco de desarrollo similar a Angular y React, para la parte front-end de la aplicación y Apex, lenguaje con sintaxis similar a Java, y SOQL para la parte de servidor (back-end).

Palabras clave: Salesforce, aplicación en la nube, Aura Components, Apex, SOQL.

Abstract

Nowadays, sales and businessmen have thousands of technological resources that help them on their work. They need to store data from their clients, suppliers, sales, etc. And they need to do it easy and efficiently.

One of the biggest technological advances in recent years has been the Cloud massive data storage. The ability to access this data in any moment, anywhere and from different devices is key to improve this ease and efficiency previously mentioned.

Therefore, this project has as its main goal developing an app in the Cloud, and meeting the basic needs of a company that has to interact constantly with the client, compile data and information in order to speed up processes that would be more time consuming in a conventional way.

In this particular case, an app for a law firm has been developed. The app allows to perform management activities with records, clients and contacts. It provides a personal agenda for each user with a calendar, events and reports.

The app has been developed on Salesforce, using the following technologies: Aura components (similar development to Angular and React) which will cover the front-end part of the app, and Apex (a language with a syntax similar to Java) and SOQL for the server part (back-end).

Keywords: Salesforce, app in the cloud, Aura Components, Apex, SOQL.

Agradecimientos

Quisiera agradecer, en un primer lugar, a mi familia y amigos por todo el apoyo y el cariño mostrado desde el principio. Gracias, por hacer de todo esto algo entretenido y llevadero.

A todos y cada uno de los profesores que he tenido durante el grado, ya que cada uno ha aportado su granito de arena y su conocimiento para poder formarme como ingeniero informático. En especial, quisiera agradecer a Patricia por acceder a ser mi directora de este trabajo, ayudándome siempre que lo he necesitado.

También quisiera agradecer a la empresa Santander Global Tech por darme la oportunidad de haber realizado las prácticas con ellos y poder abrirme paso en el mundo laboral, ya que después de mi excelente formación he pasado a formar parte del grupo Santander como empleado. En especial, a Jesús y a Pedro. Gracias por ser mis tutores allí, ayudarme, curtirme y enseñarme.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

En los últimos años la tecnología ha cambiado nuestra forma de vida, nos ha forzado a pensar y hacer las cosas de manera diferente. Tanto es así que, a día de hoy, parece impensable no recurrir a ella a la hora de hacer casi cualquier cosa.

También ha llegado a la hora de cambiar el mundo de los negocios. Ahora somos capaces de automatizar procesos, guardar enormes cantidades de información y realizar el trabajo que se hacía antaño de manera mucho más rápida y eficiente. Todo ello gracias a la gran evolución tecnológica en la que vivimos.

La mayoría de las empresas intentan buscar nuevas soluciones tecnológicas para sustituir viejas estrategias o procedimientos e implantar nuevas tecnologías que les faciliten el trabajo. Lo que se busca es, principalmente, implantar soluciones modernas que ahorren tiempo y esfuerzo. Ese es el enfoque que se seguirá en este trabajo, desarrollando una aplicación de negocio en la que se fomente el uso de herramientas y tecnologías punteras.

A lo largo de este capítulo se expondrán el contexto tecnológico, los objetivos y motivaciones de la aplicación y, finalmente, la estructura que seguirá este trabajo.

1.1 Contexto

Para ciertas empresas es muy importante almacenar y conocer datos sobre sus clientes. En especial, cuando se quiere vender un tipo de producto o servicio y se van a llevar a cabo negociaciones de por medio. Conocer ciertos detalles de tu cliente en ocasiones puede llegar a ser decisivo para llevar a cabo una venta.

Por ejemplo, en un concesionario de coches después de cerrar la venta de un coche, el vendedor ha sabido buscar la mejor oferta en base a las preferencias de su cliente. Para ello, después de cada reunión con el cliente, en vez de apuntar la información relevante en una hoja de papel, lo que ha hace es introducirlo en su aplicación en la nube, asegurándose así que la información no se va a perder, que puede ser accesible por otras personas y lo más importante, que puede ser accesible desde casi cualquier dispositivo.

Aquí es donde entra en juego la solución tecnológica que vamos a usar como base para el desarrollo de este trabajo, que es el CRM (Customer Relationship Management) [1]. Un CRM se encarga de gestionar las relaciones que tiene una empresa con sus clientes ayudando así en la gestión de ventas y ofreciendo información procesable. También se enfoca en la integración de redes sociales y correo electrónico ayudando así a la comunicación entre el equipo de trabajo.

Con esta tecnología, lo que se pretende es salvaguardar datos importantes sobre los clientes y monitorizar procesos para convertir esos pequeños detalles en información potencialmente valiosa. También es muy importante que toda esa información pueda estar disponible para otras personas ya que, por ejemplo, si una persona sale de la empresa podría darse el caso de que se lleve consigo mucha información útil a la que sólo él tenía acceso.

Algunas de las funcionalidades clave de un sistema de CRM son la gestión de contactos, la presentación de la información en forma de panel y la elaboración de informes que analizan la

actividad de los procesos. Todo ello junto con herramientas de comunicación, redes sociales, mensajería instantánea y correo electrónico integrado.

A partir de estas premisas y sabiendo que nuestra herramienta tecnológica principal va a ser un CRM, se ha de tratar de escoger la mejor opción de entre las disponibles. Hay dos tipos principales de CRM: CRM On Premise o CRM On Demand en la nube. El primero se trata de una aplicación "in-house", alojada por el cliente y gestionada por los propios empleados. La propia empresa adquiere el derecho de uso de las licencias, instala el software en sus servidores y se responsabiliza de la operación del mismo, la actualización, la seguridad de los datos y todas las actividades relacionadas con el software. La segunda opción es de tipo SaaS (Software as a Service), donde el soporte lógico y los datos se almacenan en un servidor de terceros a los que se accede a través de internet y que ofrece capacidad para la personalización de la aplicación y mantenimiento de la privacidad de los datos con los que se trabaje. El usuario paga por el uso, por la infraestructura necesaria (almacenamiento, seguridad, etc.), por el correcto funcionamiento de la aplicación y por el mantenimiento (nuevas versiones, corrección de bugs, etc). En términos económicos, la segunda solución es mejor y se adapta más a las necesidades de nuestro proyecto.

Dentro de los CRM en la nube, destaca el conocido gigante llamado Salesforce [2]. Esta herramienta es, a día de hoy, uno de los CRM más potente que existe. También es una de las herramientas más populares y usadas en el mercado por grandes y pequeñas empresas, como solución de gestor de ventas.

Figura 1.1 Logotipo de Salesforce

Desde hace unos cuantos años se están fomentando estos tipos de desarrollos en la nube ya que nos facilitan una serie de ventajas clave respecto a las tradicionales aplicaciones de cliente-servidor. Algunas de ellas son la capacidad de adaptación a cualquier tipo de empresa, ya sea grande o pequeña, ya que se dispone de muchísimos servidores conectados para almacenar millones de datos (Big Data).

1.2 Objetivos

Después de hacer un breve resumen y poner en contexto la herramienta Salesforce, se explica cómo esta herramienta puede ayudar en el desarrollo de este trabajo de fin de grado.

El objetivo que se plantea es el desarrollo de una aplicación en la nube totalmente personalizada, basada en Salesforce, que cubra las necesidades básicas de gestión de un negocio en relación a sus clientes. En concreto se he escogido realizarlo para un bufete de abogados.

Dicha aplicación cubrirá las necesidades básicas de gestión del negocio, por lo que los usuarios dispondrán de:

- Una base de datos de todos los clientes con su respectiva información, además de los expedientes que irán asociados a cada uno de ellos.

- Información sobre los contactos (otras personas que no son clientes), bufetes, etc.
- Informes que mostrarán información relevante sobre los casos y los clientes.
- Un calendario para anotar los eventos a los que han asistido o van a asistir con posibilidad de recordatorios vía email.

La aplicación podrá ejecutarse desde un navegador web en un ordenador o desde un dispositivo móvil/Tablet.

Desde el punto de vista del aprendizaje que aportará el desarrollo de este trabajo, cabe destacar que además de todo el conocimiento que se pueda adquirir como administrador de la plataforma configurando los objetos, los usuarios, la visibilidad, la seguridad, etc; también entra en juego la necesidad de aprender nuevos lenguajes de programación como Apex (Lenguaje de programación creado por Salesforce con una sintaxis similar a Java) y de programación web, utilizando Salesforce Lightning Web Components (HTML de Salesforce), JavaScript y CSS. Todo ello es necesario para adaptar la aplicación en la nube al modelo de negocio escogido.

Una vez tengamos una aplicación funcional con las necesidades básicas de negocio, y para la cual, hayamos puesto en práctica las citadas destrezas como programador, podremos concluir que los objetivos propuestos para este trabajo habrán sido conseguidos.

1.3 Estructura del documento

A lo largo de este documento se expondrán diversos capítulos donde se explicará cada una de las fases llevadas a cabo para el desarrollo de la aplicación en la nube. Los requisitos demandados por posibles usuarios, diseño de la aplicación y su implementación y prueba serán expuestos y explicados en detalle de la siguiente forma:

- En el capítulo 2 se habla sobre las herramientas y tecnologías utilizadas durante el desarrollo.
- A continuación, se recogen los requisitos funcionales y no funcionales en el capítulo 3, incluyendo casos de uso y requisitos de la interfaz de usuario.
- En el capítulo 4 se aborda el tema del diseño arquitectónico y del sistema de la aplicación en cuestión.
- En el capítulo 5 se muestra la estructura del proyecto y se expondrá la parte de implementación e integración continua.
- En el capítulo 6 se recogen las pruebas realizadas, tanto unitarias como de integración y de sistema.
- En el capítulo 7 se habla sobre las mejoras a futuro y las conclusiones del proyecto.

2. Tecnologías y herramientas

En el siguiente capítulo hablaremos sobre las herramientas, tecnologías y metodologías que se han utilizado en el desarrollo del trabajo.

2.1 Tecnologías

2.1.1 Salesforce

Salesforce es una empresa estadounidense fundada en 1999 por Marc Benioff (ejecutivo de Oracle en su momento) dedicada al desarrollo de software como servicio (SaaS). Su producto estrella es el CRM llamado Sales Cloud, aunque también tiene más productos con enfoque a la atención al cliente, marketing, inteligencia artificial, gestión de comunidades y creación de aplicativos entre otras [2].

Para proveer al usuario de este tipo de servicios, Salesforce vende estos productos como diferentes "nubes" en las cuales se desarrollan diferentes aplicaciones:

- **Sales Cloud:** enfocado al equipo de ventas. Contiene todo lo necesario para cerrar negocios, colaborar y vender en equipo, gestionar contactos y conseguir oportunidades. Como decíamos, es el producto estrella de Salesforce y es básicamente en lo que está enfocado este trabajo, haciendo uso de alguno de los objetos "core" de esta nube.
- **Service Cloud:** permite mejorar el servicio de atención al cliente de las empresas por medio de canales como el uso de herramientas para la prestación de servicios in situ, chat web, integración de telefonía (CTI) y servicio de atención al cliente en redes sociales.
- **Marketing Cloud:** contiene múltiples herramientas para llevar a cabo las tareas de marketing en la empresa. Dichas herramientas permiten gestionar de manera eficiente la interacción de la marca con sus clientes a través de diferentes canales.
- **Community Cloud:** es una infraestructura que permite la colaboración entre varias organizaciones (empleados, socios y clientes).
- **Analytics Cloud:** es la nube que está asociada con la inteligencia artificial.
- **IoT Cloud:** permite a las empresas conectar los datos de Internet of Things (IoT) así como cualquier contenido digital, con información de los clientes, todo ello en tiempo real.
- **Commerce Cloud:** herramienta de comercio electrónico B2C inteligente que ofrece al usuario todo lo que necesita para crear experiencias de compras innovadoras y personalizadas a través de diferentes dispositivos.

Algo muy positivo es que cualquier empresa que quiera o necesite cualquiera de estos servicios, los puede tener por separado o escoger los que necesite. Una vez ha contratado las licencias necesarias, de base ya tiene una aplicación operativa que puede adaptar a su modelo de negocio según sus necesidades. Todo depende de la personalización que se quiera dar.

2.1.2 Lightning Aura Components

Aura [3] es un framework de código abierto que se creó en 2013 para desarrollar aplicaciones móviles y de escritorio dentro de la plataforma de Salesforce. Permite crear aplicaciones que son independientes de los datos que residen en la plataforma.

Los componentes de Aura son unidades únicas y reutilizables de una aplicación. Representan una sección reutilizable de la interfaz de usuario y pueden variar en granularidad desde una sola línea de texto hasta una aplicación completa. Además, no necesitan optimizarse por dispositivo y ofrecen responsividad.

El marco incluye un conjunto de componentes prediseñados. Por ejemplo, los componentes que vienen con el estilo Lightning Design System están disponibles en el espacio de nombres Lightning. Estos componentes también se conocen como componentes Lightning base. Se pueden ensamblar y configurar componentes para formar nuevos componentes en una aplicación. Los componentes se renderizan para producir elementos HTML dentro del navegador.

Un componente Aura puede contener otros componentes, así como HTML, CSS, JavaScript o cualquier otro código de programación web. Esto permite crear aplicaciones con interfaces de usuario complejas. Los detalles de la implementación de un componente están encapsulados. Esto ayuda a que el consumidor de un componente se centre en crear su aplicación, mientras que el autor del componente puede innovar y realizar cambios sin perjudicar a los consumidores. Los componentes se configuran estableciendo los atributos con nombre que exponen en su definición e interactúan con su entorno escuchando o publicando eventos.

El componente como entidad se dividiría en tres capas a nivel de programación. La primera sería la parte de Aura, donde se pueden usar etiquetas HTML o las propias de Aura. La segunda capa sería el controlador, la parte de JavaScript, que normalmente suele estar compuesta por dos archivos ".js". El primero tendría las llamadas a todas las funciones y el segundo sería un archivo que guardaría funciones de apoyo. Por último, tenemos la parte de los estilos, que se guardarían en un archivo ".css".

2.1.3 Apex

Apex es un lenguaje de programación orientado a objetos con una sintaxis similar a Java. Permite ejecutar declaraciones de control de transacciones y flujos en el servidor de la plataforma de Salesforce, junto con llamadas a la API de la plataforma Lightning [4].

El código Apex puede ser utilizado de dos formas diferentes:

- En una **clase**, que al igual que en Java, se trata de una plantilla o estructura de la cual se crean objetos Apex. Las clases contienen métodos, variables, llamadas a otras clases, etc.
- En un **desencadenador** (Trigger). Es código que se ejecuta cada vez que se realiza una inserción, modificación o borrado de un objeto en la plataforma, es decir, una operación CRUD.

Una vez se ha desarrollado el código de una clase, para poder pasar a un entorno productivo, necesita tener obligatoriamente una clase de test que cubra, al menos, el 75% de sus líneas de código.

2.1.4 SOQL

SOQL [5] al igual que SQL es un lenguaje de dominio específico que utiliza Salesforce para recuperar la información de los registros guardados en su base de datos relacional. Las sentencias se pueden ejecutar directamente desde el lenguaje Apex, lo cual hace bastante fácil la programación a la hora de hacer consultas, inserciones, modificaciones o borrados.

2.2 Herramientas

2.2.1 Visual Studio Code

Para el desarrollo del trabajo se ha utilizado la versión de escritorio del editor de código Visual Studio Code. Por defecto, viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes (como C++, C#, Java, Python, PHP, Go, Apex) y runtimes (como .NET y Unity).

Al desarrollar código para Salesforce, hay una serie de extensiones que son esenciales para el programador y son las siguientes [6]:

- **Salesforce Extension pack:** trae consigo 6 extensiones.
    - Salesforce CLI Integration: esta extensión se vincula principalmente con Salesforce CLI y permite conectarse con el entorno de Salesforce mediante los comandos de Salesforce DX.
    - Apex: extensión de la sintaxis del código.
    - Apex Interactive Debugger: esta extensión permite depurar el código de Apex directamente desde el Visual Studio Code.
    - Apex Replay Debugger.
    - Lightning Web Components: permite crear, editar e implementar componentes web Lightning directamente desde el Visual Studio Code.
    - Aura Components: permite trabajar con Lightning components normales que utilizan el marco de Aura.
    - Visualforce: permite trabajar con páginas y componentes de Visualforce directamente desde el Visual Studio Code.
- **Apex PMD:** esta extensión está relacionada con la calidad de código. Puede analizar tu código en tiempo real y resaltar los fallos o simplemente se puede configurar para que el usuario lo ejecute cuando quiera. Es muy útil, ya que no se fija en los fallos de sintaxis (tu código puede funcionar y compilar perfectamente), sino en cosas como la complejidad ciclomática o en reglas que el usuario mismo puede haber configurado.
- **ES Lint:** sirve para corregir errores de sintaxis de JavaScript.

2.2.2 Git

El sistema de control de versiones Git se ha utilizado para gestionar las diferentes versiones del proyecto a lo largo de su desarrollo. Esta herramienta permite mantener todo el desarrollo en un lugar online y poder acceder a él desde cualquier máquina que tenga internet.

El IDE VSCode tiene una integración con esta herramienta, lo que permite tener siempre una actualización con los cambios en un repositorio y poder trabajar de forma paralela favoreciendo así un modelo de integración continua, en el que se profundizará más adelante.

Esta herramienta se ha convertido en un standard dentro del desarrollo de proyectos gracias al uso con comandos sencillos, la posibilidad de trabajar de manera offline sin tener que estar conectado a un sistema central y su facilidad para revertir cambios.

2.2.3 Developer Console

Se trata de una herramienta online que proporciona Salesforce para el desarrollador. Va ligada al entorno de trabajo y permite la creación y modificación de las aplicaciones, componentes, código Apex, consultas, ejecuciones de clases de test, etc. Es una herramienta bastante completa. Sin embargo, solo permite trabajar de forma individual y los cambios realizados en

la consola del desarrollador producen cambios en el entorno in situ, lo cual puede dificultar la programación paralela de varios desarrolladores.

2.2.4 Dataloader

Es una herramienta con la que se pueden realizar operaciones CRUD (insertar, modificar, extraer o eliminar) sobre los registros de Salesforce de forma masiva. El usuario debe introducir las credenciales de usuario de su entorno, indicar qué tipo de entorno es (entorno de prueba, desarrollo, de producción, etc.) y un archivo ".csv" que contenga los registros. La herramienta permite realizar operaciones de hasta 5 millones de registros, donde se pueden configurar el número de registros del batch, etc.

2.3 Metodología

Para llevar a cabo el desarrollo del software es necesario seguir una estrategia que permita llevar un orden y seguir unas pautas. Con ello, la calidad del producto a un coste razonable en tiempo y esfuerzo está asegurada. Esa estrategia se llama metodología de desarrollo del software [7]. Las metodologías que tradicionalmente se usan son las siguientes: modelo en cascada, prototipado, incremental, espiral y RAD (Rapid Application Development). Sin embargo, a principios de este siglo y sobre todo en los últimos años ha aparecido un nuevo modelo conocido como "Agile", el cual se ha implantado en la mayoría de empresas. Hay varios modelos de metodologías ágiles, en concreto, para este proyecto se ha utilizado una metodología ágil de tipo Kanban [8].

La base principal de una metodología Kanban es la puesta de todas las tareas en un tablero. Dicho tablero se divide en columnas que identifican el estado de cada tarea, de forma que, al inicio aparecen todas las tareas en una columna ordenadas por orden de prioridad, como en una cola, y se van repartiendo las historias y avanzándolas a la columna de desarrollo. Una vez vayan pasando todas las fases, acabarán en la última columna que significa que el trabajo ya está hecho y entregado al cliente.

Para este trabajo, se ha utilizado la herramienta Trello. Esta herramienta nos permite poner en práctica la metodología ágil de tipo Kanban, como se puede apreciar en la figura 2.1 que se muestra a continuación.

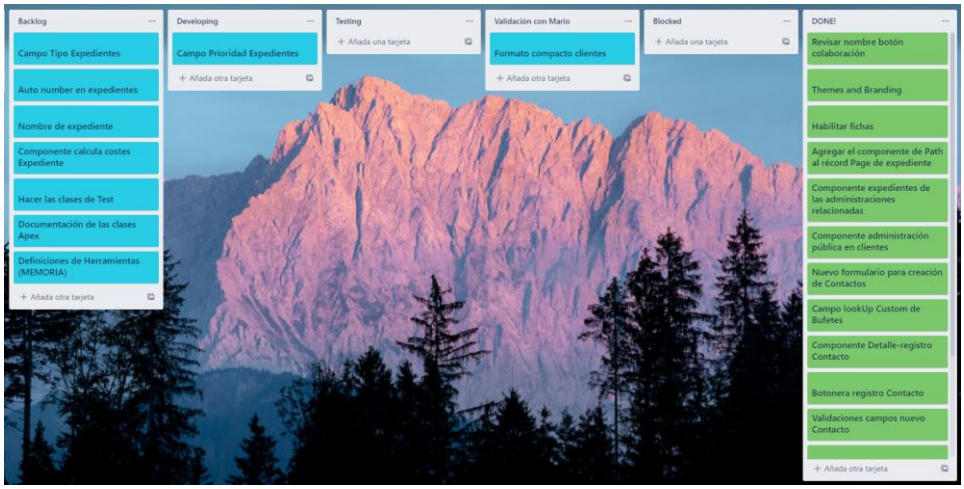


Figura.2.1 Tablero Kaban

3. Especificación de requisitos

A lo largo de este capítulo se describe el conjunto de requisitos funcionales y no funcionales, así como los casos de uso y requisitos de interfaz que se han tenido en cuenta para el desarrollo del proyecto.

Además de la investigación personal para recoger los requisitos de la aplicación, se han mantenido conversaciones con una persona conocedora de la materia, con estudios universitarios y de postgrado que ha ejercido la profesión de abogado. Así, podemos garantizar que se dispone de una funcionalidad óptima, que cumple con los requisitos y exigencias, y que además, es totalmente operativa para el modelo de negocio escogido.

3.1 Especificación de requisitos funcionales

Cuando se habla de requisitos funcionales se hace referencia a todas aquellas funciones o servicios que, en su conjunto, responden a las interacciones que se llevan a cabo entre el usuario y la aplicación, aunque también se incluyen la interacción con otros sistemas, respuestas automáticas o procesos predefinidos.

En la siguiente tabla se muestran los requisitos funcionales recogidos para este proyecto.

Tabla 3.1 Requisitos funcionales

Identificador	Descripción
1	El usuario debe poder gestionar expedientes.
1.1	El usuario debe poder añadir / eliminar / modificar sus expedientes.
1.2	El usuario debe poder filtrar por número o por asunto sus expedientes y los que le compartan los demás usuarios.
1.3	El usuario debe poder obtener los costes y la facturación por cada expediente. Además podrá mandar por correo dichos costes directamente al cliente, utilizando una plantilla de email.
1.4	El usuario debe poder relacionar sus expedientes con los clientes, contactos, persona contraria y otros usuarios del sistema.
1.5	El usuario debe poder compartir sus expedientes con otros usuarios.
1.6	El usuario debe poder cambiar el estado del expediente.
2	El usuario debe poder gestionar clientes.
2.1	El usuario debe poder añadir / eliminar / modificar sus clientes.
2.2	El usuario debe poder distinguir entre los siguientes tipos de cliente: administración pública, persona física y persona jurídica.
2.3	El usuario debe poder relacionar las personas físicas que representen a una administración pública o a una persona jurídica.
2.4	El usuario debe poder compartir sus clientes con otros usuarios.
3	El usuario debe poder gestionar contactos.

3.1	El usuario debe poder añadir / eliminar / modificar sus contactos
3.2	El usuario debe poder distinguir entre tres tipos de contacto: procurador, abogado de la otra parte o perito. Los contactos de tipo abogado de la otra parte se pueden relacionar con bufetes.
3.3	El usuario debe poder compartir sus contactos con otros usuarios.
4	El usuario debe poder gestionar eventos
4.1	El usuario debe poder añadir / eliminar / modificar / consultar eventos dentro de un calendario.
4.2	El usuario debe poder recibir recordatorios vía email de los eventos programados.
5	El usuario debe poder generar reportes con estadísticas en base a sus registros de clientes y expedientes.
5.1	El usuario debe poder visualizar los reportes creados.

3.2 Especificación de requisitos no funcionales

Los requisitos no funcionales son aquellos aspectos que son indispensables o que se han tenido en cuenta para el correcto funcionamiento de la aplicación que no describen información a guardar o funciones específicas a realizar, sino características de funcionamiento.

Los requisitos no funcionales capturados para este proyecto se muestran en la siguiente tabla.

Tabla 3.2 Requisitos no funcionales

Identificador	Descripción	Categoría
1	Debe establecerse una jerarquía de roles y perfiles dentro de la aplicación. Tener un perfil diferente significa disponer de diferentes permisos dentro de la plataforma sobre los objetos y las diferentes funcionalidades. Tener un rol diferente significa poder acceder a diferentes registros dependiendo de la jerarquía de tu rol en el sistema.	Seguridad
2	Se ha de implementar autentificación de dos factores (2FA). La configuración de dos factores consiste en verificar que el usuario que intenta acceder a la aplicación es quien dice ser. Para ello, el usuario suele introducir un código que se envía a un dispositivo que posee y que ha sido previamente registrado.	Seguridad
3	La aplicación deberá ser accesible desde el navegador y en la aplicación móvil de Salesforce.	Portabilidad
4	La aplicación estará disponible en inglés y español.	Usabilidad

3.3 Especificación de casos de uso

En este apartado se explican en detalle los casos de uso de la aplicación. Los casos de uso se encargan de recoger las principales funcionalidades a alto nivel, que a su vez, se exponen en diagramas que muestran una idea básica de lo que se quiere hacer.

Primero, se ofrece un diagrama general en la figura 3.1 que engloba todos los casos de uso de alto nivel de la aplicación.

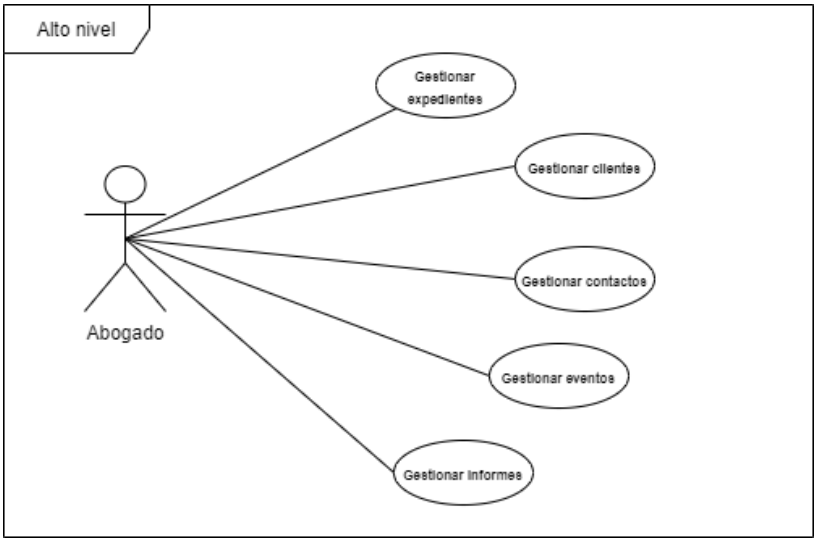


Figura 3.1 Diagrama de casos de uso de alto nivel

La figura 3.2 detalla las funcionalidades englobadas en la gestión de expedientes.

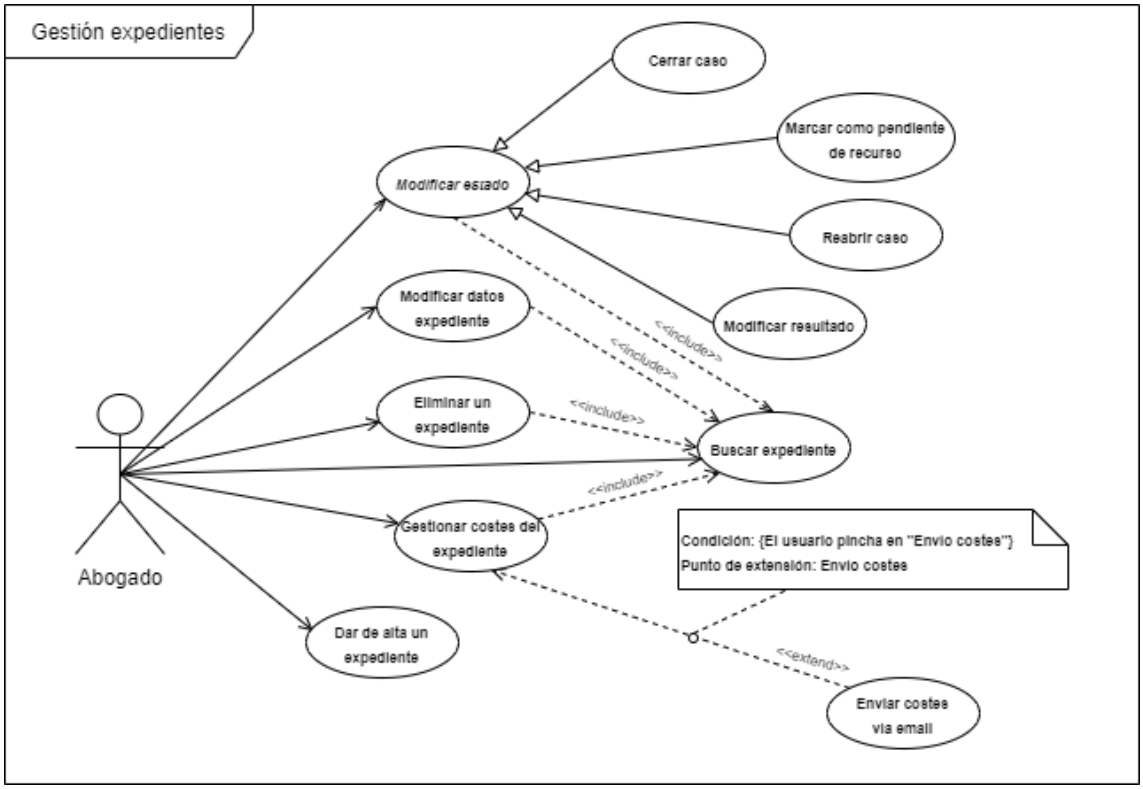


Figura 3.2 Diagrama de casos de uso de gestión de expedientes

A continuación, también se incluye los diagramas de casos de uso detallados correspondientes a la gestión de contactos y clientes, así como la gestión del calendario y la generación de reportes, en las figuras 3.3, 3.4, 3.5 y 3.6, respectivamente.

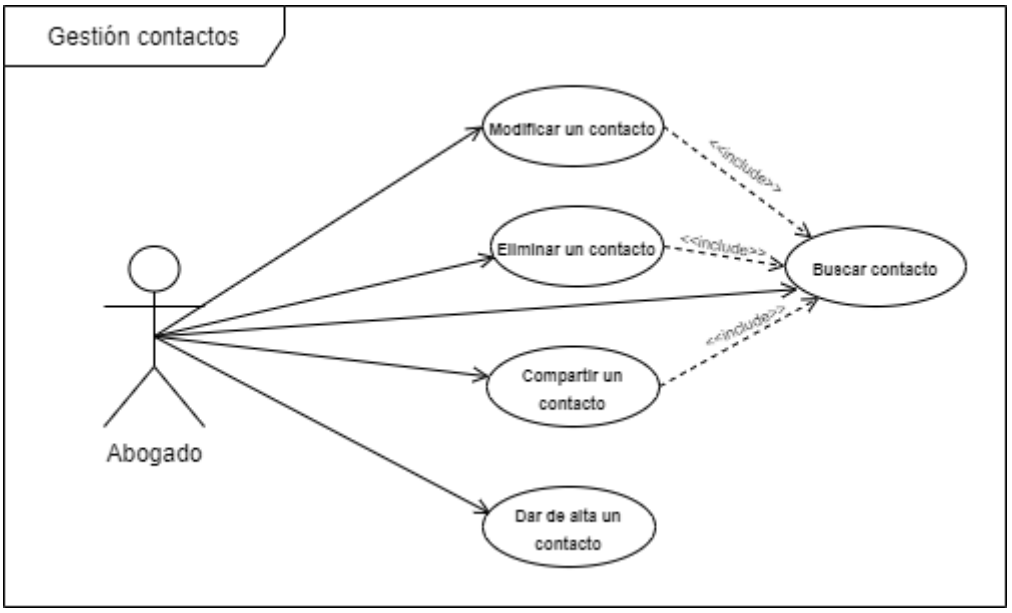


Figura 3.3 Diagrama de casos de uso de gestión de contactos

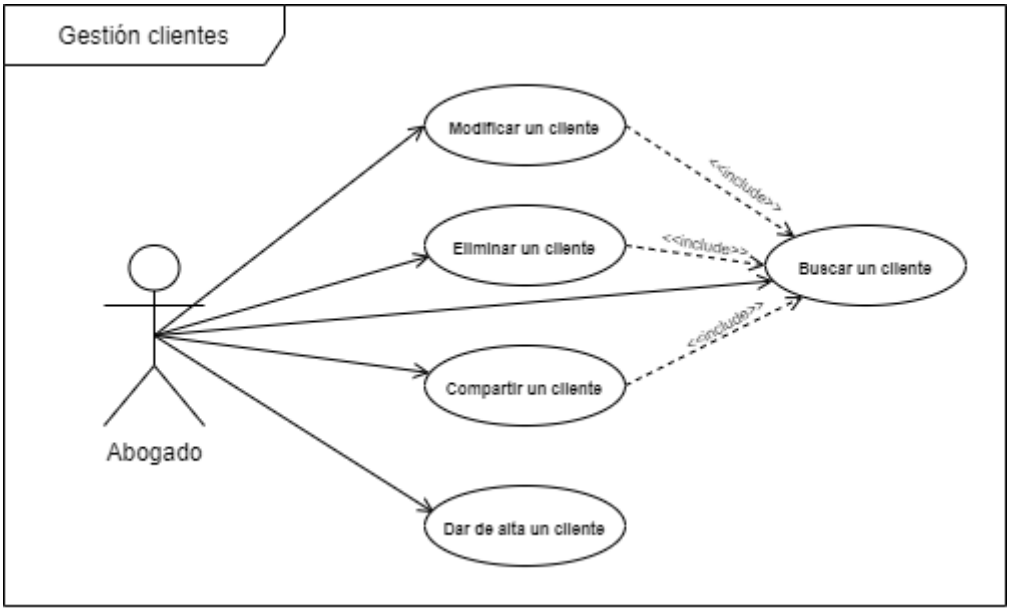


Figura 3.4 Diagrama de casos de uso de gestión de clientes

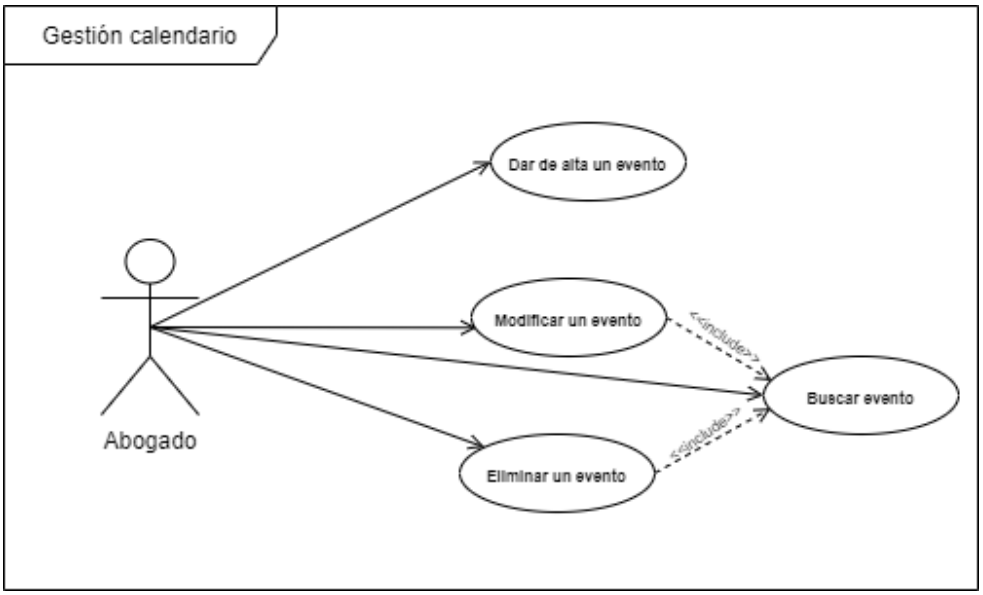


Figura 3.5 Diagrama de casos de uso de gestión de eventos del calendario

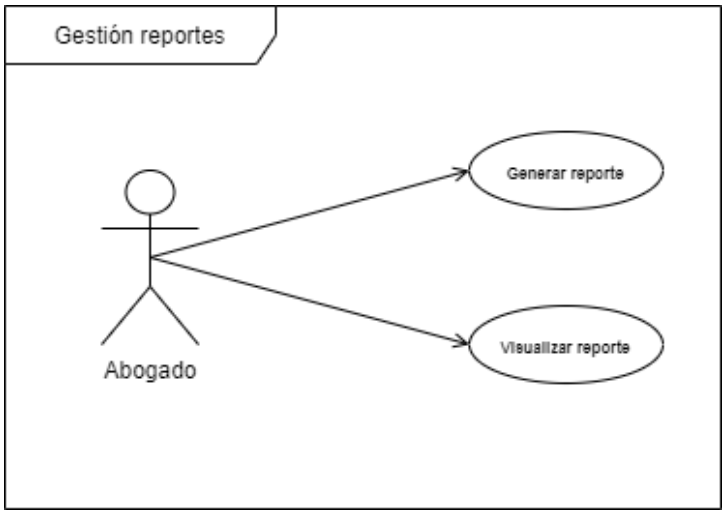


Figura 3.6 Diagrama de casos de uso de gestión de reportes

Con los diagramas, se ve a grandes rasgos las funcionalidades y casuísticas que se quieren implementar. Para entrar un poco más en detalle, se especificarán algunos de los casos de uso mediante una plantilla. Para ello, se han seleccionado algunos de los casos de uso recogidos en los diagramas anteriores, cuya especificación se recoge en las tablas 3.3, 3.4 y 3.5.

Tabla 3.3 Búsqueda de un expediente

Identificador	UC-1.2
Nombre	Buscar expediente.
Descripción	El usuario realiza la búsqueda de un expediente.
Actor/es	Abogado.
Precondiciones	El expediente tiene que estar creado.
Flujo Principal	1. El usuario selecciona la pestaña de expedientes. 2. El usuario introduce el número/asunto del expediente en el

	buscador. 3. El sistema busca el expediente. 4. El sistema muestra los detalles del expediente.
Postcondiciones	Se ha mostrado el expediente.
Flujos alternativos	3.a. No existe un expediente con los datos introducidos por el usuario. 3.a.1 El sistema informa de que el expediente no existe.

Tabla 3.4 Generar costes de un expediente

Identificador	UC-1.3
Nombre	Gestionar costes expediente.
Descripción	El usuario quiere registrar los costes del expediente.
Actor/es	Abogado.
Precondiciones	El expediente tiene que estar creado.
Flujo Principal	1. Include Buscar expediente. 2. Dentro del registro del expediente, el usuario selecciona la opción de editar en el formulario de costes. 3. El usuario rellena los datos. 4. El usuario selecciona la opción de guardar. 5. El sistema calcula, almacena y muestra los costes totales. 6. Extension point: Envío costes.
Postcondiciones	Los costes del expediente han sido calculados y registrados.
Flujos alternativos	En los puntos 1, 2 y 3, el usuario puede seleccionar la acción de cancelar.

Tabla 3.5 Enviar costes de expediente

Identificador	UC-1.3
Nombre	Enviar costes expediente.
Descripción	El usuario quiere enviar los costes del expediente a un cliente.
Actor/es	Abogado.
Evento de activación	El caso de uso comienza cuando se cumple la condición del punto de extensión “Envío costes” del caso de uso “Generar costes expediente”.
Precondiciones	Los costes del expediente tienen que estar registrados.
Flujo Principal	1. El sistema muestra un texto que indica si de verdad quiere enviar los costes del expediente a la dirección de correo electrónico del cliente. 2. El usuario selecciona la opción de aceptar. 3. El sistema envía un correo electrónico al cliente con la información sobre los costes. El sistema muestra una ventana en la cual se indica que el correo ha sido satisfactoriamente enviado al cliente.
Postcondiciones	El cliente ha sido notificado de los costes del expediente.
Flujos alternativos	El usuario le puede dar al botón de cancelar en cualquier momento. 2.a El usuario ya había enviado los costes previamente al cliente. 2.a.1 El sistema avisa al usuario de que ya ha mandado los costes del expediente, al menos, una vez. 3.a El mensaje no se envía. 3.a.1 El sistema informa al usuario que el mensaje no se ha podido enviar.

3.4 Requisitos de interfaz

Por otro lado, además de los requisitos de usuario mencionados, se han establecido unos requisitos de interfaz para hacer que la aplicación sea lo más sencilla y funcional posible de cara al usuario final. Proporcionar un software que sea muy complejo de entender para el usuario puede acarrear problemas, ya que el usuario se puede cansar de tener que realizar siempre los mismos pasos costosos para acometer las tareas del día a día.

Es por ello que tanto la versión del navegador como la aplicación móvil disponen de una interfaz fácil y sencilla de entender. La figura 3.7 muestra a modo de ejemplo, un mockup de la página principal de la versión de navegador. Cómo se puede apreciar, se usará una barra horizontal que contendrá las diferentes secciones de la aplicación a modo de menú. A partir de cada una de las opciones, el usuario podrá interactuar con las diferentes funcionalidades previamente mencionadas. La figura 3.8, por su parte, presenta un mockup de la página principal en la versión móvil.

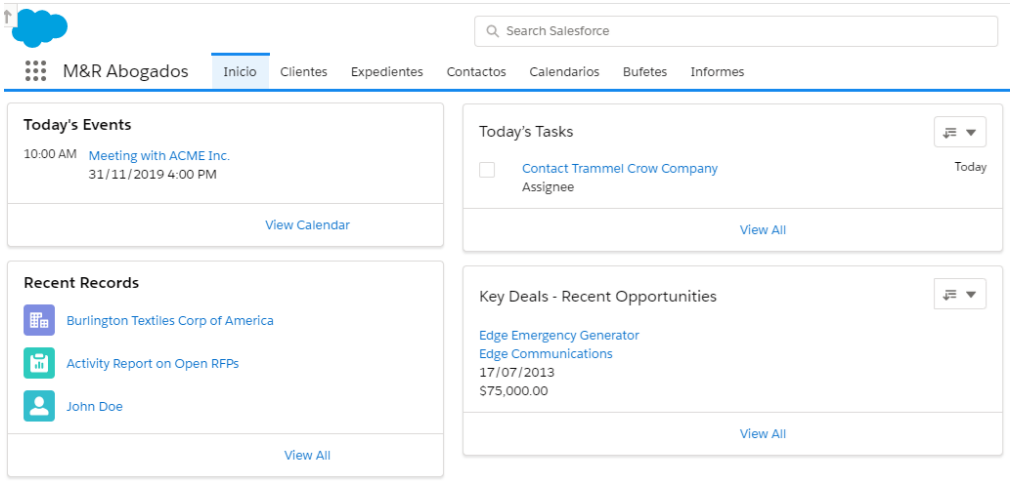


Figura 3.7 Página principal versión de navegador

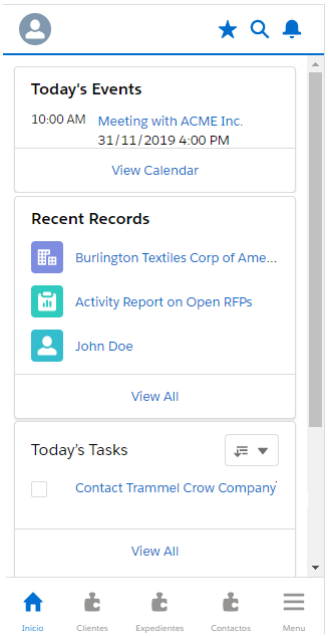


Figura 3.8 Página principal versión móvil

4. Diseño software

Para cumplir con el análisis previo, es necesario llevar a cabo un buen diseño del sistema que permita cumplir con los requisitos mencionados anteriormente. Con el paso del tiempo, se han establecido una serie de reglas y directrices para llevar a cabo esta labor. Éstas, en su conjunto, hacen que la producción del mismo sea más eficaz al hacer uso de unos patrones de diseño.

En este capítulo se explican qué pasos se han seguido para llevar a cabo la fase del diseño del software. La parte del diseño se divide en dos partes: diseño arquitectónico y diseño detallado.

4.1 Diseño arquitectónico

En ingeniería del software, cuando se habla de diseño arquitectónico, se refiere a lo que comúnmente se conoce como el diseño de alto nivel o arquitectura de la aplicación [7]. Para ello, se definen todos los componentes que van a formar parte del sistema y se establece la forma en la que van a interactuar los unos con los otros, así como donde se desplegará cada uno de ellos.

4.1.1 Patrón arquitectónico de la aplicación

Es necesario definir una propuesta de diseño que satisfaga con éxito y solvencia todas las propuestas de requisitos funcionales y no funcionales. Para ello, no se tiene que definir una estrategia nueva, si no que ya hay varias estrategias de diseño de arquitectura software definidas que ayudarán a que se complete este proceso.

Para el desarrollo de este proyecto se ha utilizado un patrón de diseño de tipo modelo-vista-controlador (MVC). Este patrón consiste en separar en tres partes los diferentes componentes que, en su conjunto, formarán el software de la aplicación. El modelo MVC aplicado a Salesforce se puede ver definido en la siguiente figura 4.1, donde se distribuyen los componentes dependiendo de a qué capa pertenezcan.

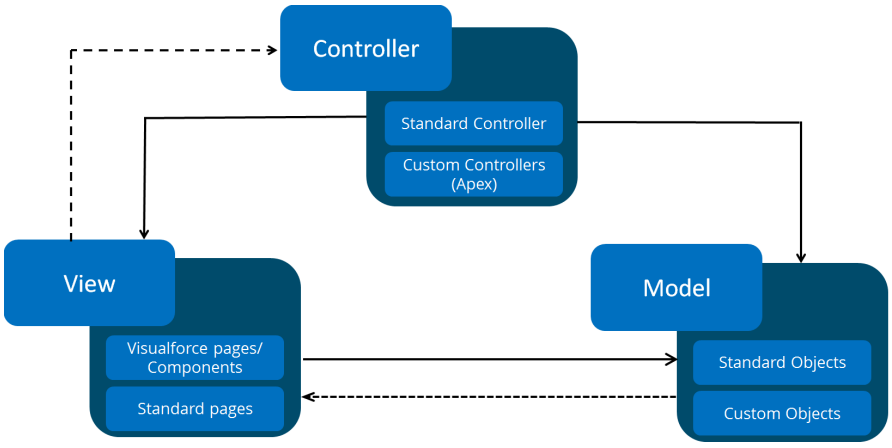


Figura 4.1 MVC aplicado a Salesforce [9]

Para comprender en que consiste, a continuación también se detalla el funcionamiento de cada capa:

- Modelo: contiene los datos de la aplicación. Se encarga de gestionar el almacenamiento y la forma en la que se guardan los datos. En Salesforce existe la entidad “Objeto”, que hace referencia a una entidad que forma parte de la aplicación. Por defecto, existen unos

objetos estándar que hacen referencia a las ventas y a los clientes. Por otro lado, también están los objetos personalizables, que nos permiten adecuar nuestro modelo de datos a cualquier dominio.

- Controlador: es la capa que conecta la vista y el modelo. Además, se encarga de gestionar la lógica y el funcionamiento de los diferentes procesos de la aplicación. Las clases Apex desempeñarán este trabajo.
- Vista: es la parte visual dirigida al usuario final. En esta capa el usuario puede interaccionar con la aplicación y hacer uso de la misma. Las páginas estándar, los componentes Lightning y las páginas Visualforce dan soporte a esta parte.

4.1.2 Arquitectura de la aplicación

Una vez definido el patrón a aplicar en el diseño de la aplicación, es conveniente hacer uso de un diagrama de componentes para poder mostrar así el resultado de la arquitectura de la aplicación. Con esto, se tiene una representación clara de las funciones y las interacciones de los componentes de las diferentes capas entre sí.

En la figura 4.2 se muestra el diagrama donde cada uno de los componentes hace referencia a los casos de uso previamente vistos en el capítulo anterior. Los componentes muestran el comportamiento lógico de negocio de la aplicación:

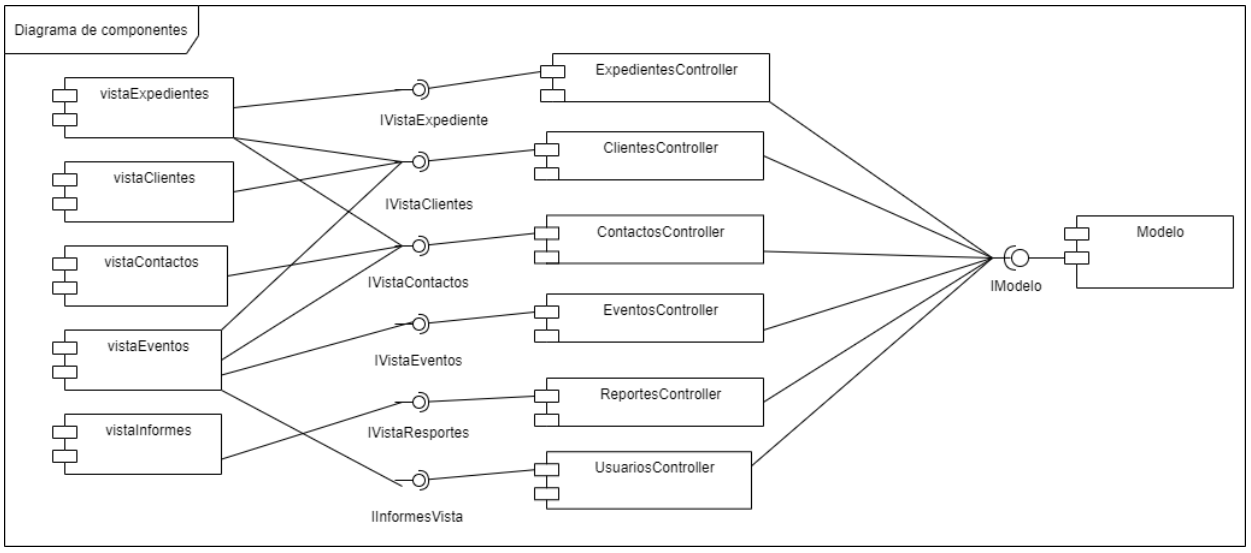


Figura 4.2 Diagrama de componentes

4.2 Diseño detallado

El diseño detallado, como su propio nombre indica, consiste en extender el detalle del diseño hasta llegar a los componentes o módulos más simples que lo conforman (las clases, en el caso de una aplicación orientada a objetos). Es, junto con el diseño arquitectónico, la base para la construcción del proyecto.

4.2.1 Clases

En este apartado se muestra el diagrama de clases del proyecto, el cual se puede ver a continuación en la figura 4.3. Un diagrama de clases nos da una visión gráfica de cómo está estructurado el sistema, viendo las dependencias y las interacciones de las diferentes entidades que lo forman.

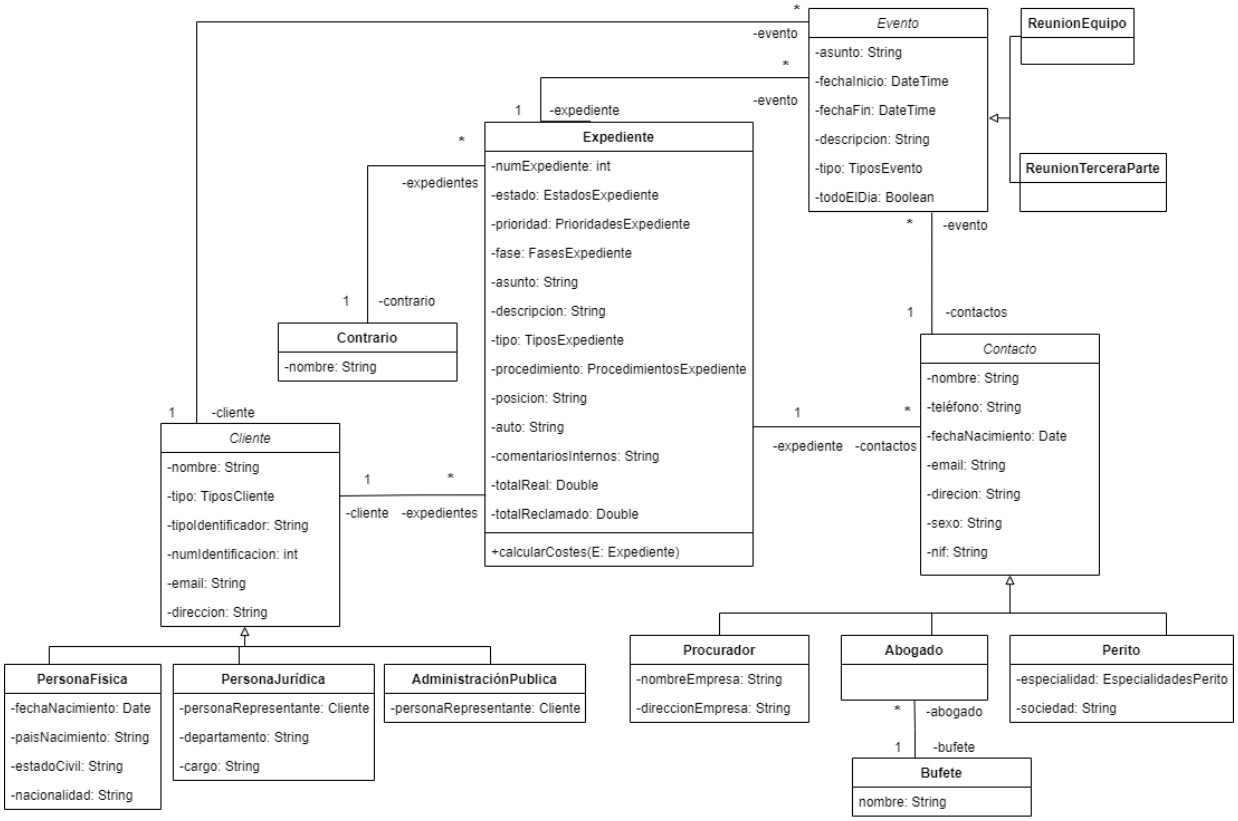


Figura 4.3 Diagrama de clases

4.2.2 Diagrama de base de datos

Por otro lado, el diagrama de base de datos permite tener una visión de cómo se van a almacenar los registros en la aplicación, qué atributos o campos van a tener y las relaciones que van a tener entre sí. Además, puede ayudar a futuro a realizar consultas sobre datos de la aplicación.

Dentro del entorno de Salesforce donde se ha desplegado la aplicación, existe una herramienta llamada “Generador de esquemas” que permite realizar un esquema personalizable de las tablas existentes en la organización. Lo bueno de esta herramienta es que permite directamente la creación de campos o relaciones en base de datos. La figura 4.4 que se muestra a continuación representa las tablas con sus atributos, los cuales se han utilizado para el desarrollo de la aplicación.

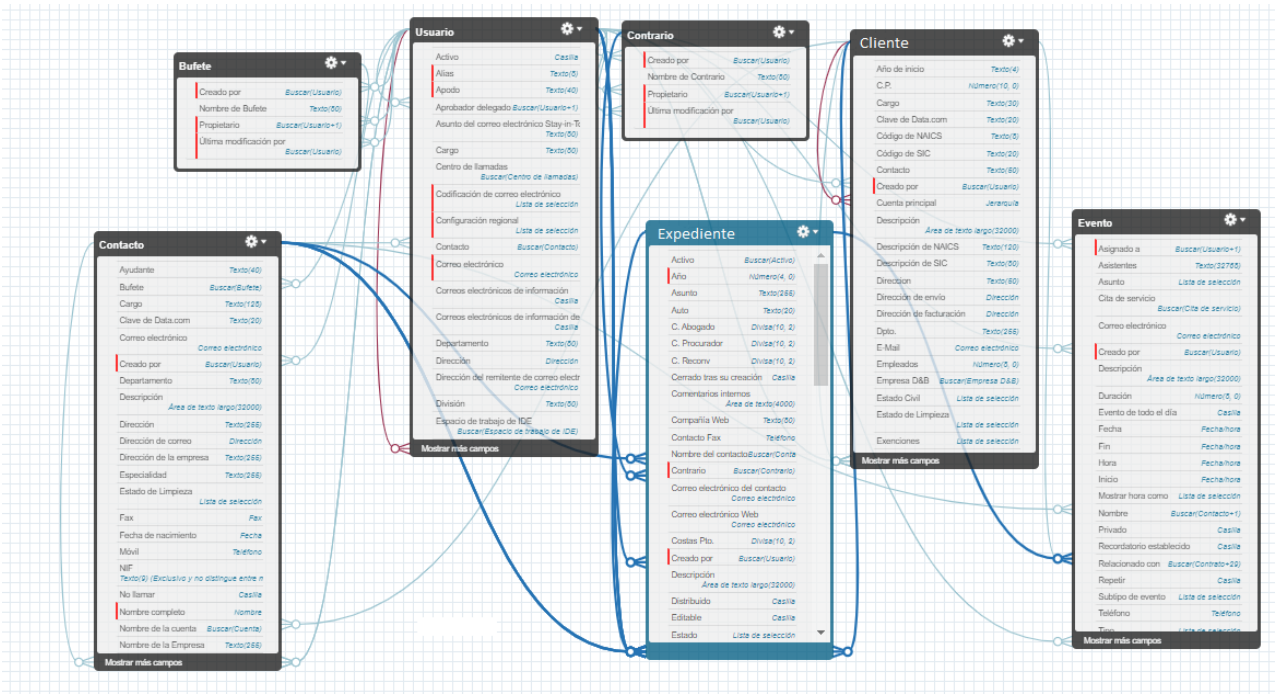


Figura 4.4 Diagrama de BBDD

5. Implementación

Una vez definidas y finalizadas las fases de diseño y arquitectura, en este capítulo se exponen las actividades que se han llevado a cabo en la fase de implementación del proyecto.

Se mostrará cómo está estructurada y desarrollada la aplicación, así como el funcionamiento de algunas de las herramientas utilizadas que han sido definidas en el capítulo dos.

5.1 Estructura del proyecto

Para el desarrollo de código, se ha usado tanto el IDE VSCode como la herramienta integrada de Salesforce para desarrolladores Developer Console.

El uso de ambas herramientas es importante para el programador, ya que cada una tiene sus ventajas y desventajas, que quedan reflejadas en la siguiente tabla 5.1:

Tabla 5.1 Visual Studio Code vs Developer Console

Herramienta	Ventajas	Desventajas
Visual Studio Code	• Visualización del proyecto estructurado por entidades. • Uso de plugin de ayuda al desarrollador como Apex pmd y ES Lint. • Sincronización con Git. • Desarrollo en local. • Sincronización entre desarrolladores.	• Necesidad de despliegue cada vez que se necesita probar cualquier cosa.
Developer Console	• Cambios directos. Cuando se guarda cualquier cambio se puede probar directamente en el entorno sin necesidad de despliegue. • Consultas SOQL directas. • Ejecución de clases de Test.	• Dificultad para el desarrollo de dos programadores a la vez.

Combinando las ventajas de cada una de estas dos herramientas, se pueden llevar a cabo buenas prácticas de desarrollo, utilizándolas adecuadamente durante el proceso de implementación.

Dentro de la herramienta VSCode se puede ver cómo está organizado el proyecto, desde las clases Apex hasta los componentes Lightning y los objetos del sistema. Todo ello, lo vemos representado por la figura 5.1 y se explica a través de las siguientes subsecciones.

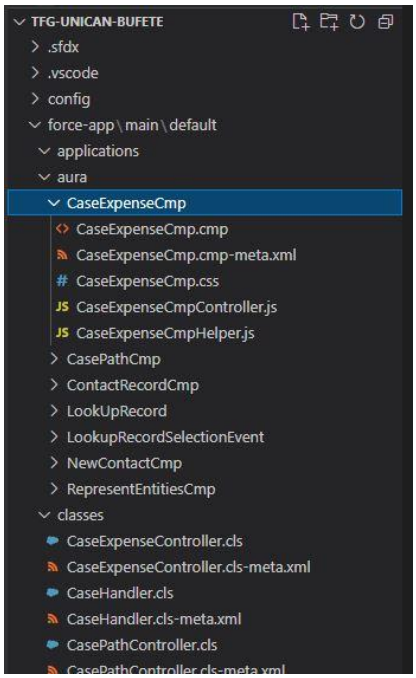


Figura 5.1 Estructura del proyecto

5.1.1 Componentes Aura

Los componentes Aura vienen definidos en subpaquetes del paquete *aura*, que a su vez, contienen varios archivos que forman el componente en su conjunto. Para explicar cómo funciona cada elemento que conforma un componente, se tomará como ejemplo el componente *CaseExpenseCmp* que aparece en la figura 5.1 del apartado anterior. Los archivos que lo implementan son:

- *CaseExpenseCmp.cmp*: contiene la parte de los elementos visuales del componente. Está definido en base a etiquetas Aura aunque también se pueden utilizar etiquetas HTML.
- *CaseExpenseCmpController.js*: contiene las diferentes funciones que son invocadas desde la vista. El código de programación utilizado es JavaScript. Su función consiste en procesar los datos y hacer funcionar la lógica del componente. También se encarga de realizar la conexión con las clases Apex.
- *CaseExpenseCmpHelper.js*: contiene funciones auxiliares que son llamadas desde el correspondiente archivo *CaseExpenseCmpController.js*.
- *CaseExpense.css*: contiene los estilos de la vista. Al igual que en HTML, se pueden crear diferentes clases con estilos personalizados. Estas clases son asignadas en las etiquetas Aura o HTML.

5.1.2 Triggers

Existe un paquete llamado *triggers* que contiene el código de los Triggers del sistema. Cada Trigger es implementado por una clase Apex y se corresponde con una de las entidades de la aplicación, por lo que serán invocados cada vez que se realice una operación CRUD sobre un objeto de dicha entidad. Tal y como se explicar en el siguiente apartado, la lógica consecuente se ejecuta en las clases *util*.

5.1.3 Clases

Dentro del paquete *classes* están las diferentes clases Apex, de las cuales se puede hacer una distinción según su uso:

- *CaseExpenseController.cls*: se utiliza la terminación *Controller* para indicar que la clase Apex va asociada a un componente, y que por tanto, tendrá la lógica, métodos y consultas a base de datos necesarias para dar soporte a la interacción entre la vista y el modelo de la aplicación.
- *CaseHandler.cls* y *CaseUtil.cls*: corresponden a la lógica que se dispara en los triggers. En este caso, la entidad asociada es Expediente (Case). Cuando se realiza una operación de inserción, modificación o eliminación de un registro esta lógica se activa. El Trigger realiza la llamada a la clase *CaseHandler.cls* en el que hay una sentencia lógica *switch* que, dependiendo de la operación invocará a los métodos correspondientes de la clase *CaseUtil.cls*. Por ejemplo, al crear un expediente sin rellenar el número de expediente, este se rellenará automáticamente.
- *CaseExpenseControllerTest.cls*: para las clases de prueba, se utiliza el sufijo *Test*. Además, para distinguir una clase de prueba es necesario añadir una anotación *@isTest* en la definición de la clase para que el sistema las identifique.

5.1.4 Objetos

También existe un paquete que guarda los metadatos de los objetos de la aplicación. Éstos son archivos XML formados por la configuración de cada objeto. Se pueden modificar tanto los objetos custom como los estándares de Salesforce.

Para cado objeto pueden modificarse múltiples aspectos, como por ejemplo los campos, las reglas de validación, la vista en listas personalizadas, el tipo de registro, etc.

5.2 Despliegue del proyecto

El código del proyecto se desarrolla de forma local, en un dispositivo que tenga la herramienta VSCode y acceso a internet, para poder realizar los correspondientes despliegues. Cada vez que realizamos un cambio en el código de forma local, para comprobar su funcionalidad se tiene que realizar un despliegue en el entorno. Así mismo, cuando se realice un cambio directamente en la configuración del entorno, tanto en la Developer Console como en la configuración de algún objeto, tenemos que actualizar dichos cambios en la versión local, para así poder subirlos a la rama máster del proyecto en Git.

Para poder hacer un despliegue, es necesario tener instalado Salesforce CLI. Salesforce CLI es una interfaz de línea de comandos que nos va a permitir trabajar con la organización de Salesforce. Los comandos pueden ser ejecutados desde la terminal o con los plugins de Salesforce para VSCode mencionados en el capítulo dos, haciendo clic derecho sobre la entidad que se quiera desplegar y seleccionando una opción llamada “SFDX: Deploy source to Org”. Algunos de los comandos más usados son los siguientes [10]:

- Desplegar recurso: sfdx force:source:deploy -p path/to/source
- Recuperar recurso: sfdx force:source:retrieve -p path/to/source
- Eliminar recurso: sfdx force:source:delete -p path/to/source

Trabajar de forma local ofrece grandes ventajas, como se ha visto en la tabla 5.1, pero además es imprescindible añadir una herramienta que controle los cambios y en la que se almacenen las diferentes versiones. Para ello, a todo esto que se ha visto se le añade el uso de la

herramienta Git y la forma de trabajar aplicando integración continua que se explicará en el siguiente apartado.

5.3 Integración continua

Una vez el proyecto está en marcha y se han completado las fases de análisis y de toma de requisitos es muy importante, antes de empezar la fase de implementación, el tener definida una estrategia para llevar a cabo los desarrollos. Es por ello que a día de hoy, se hace uso de herramientas de control de versiones como Git, subversión (SVN), etc. Estas tecnologías permiten llevar un control sobre el código y los cambios desarrollados de la aplicación.

Cuando se trabaja sobre un proyecto real, no se programa directamente en la aplicación o en el entorno que utilizan los usuarios. Tiene que haber unos entornos previos por los que pase el código desde que se desarrolla, hasta que recae en el usuario final, pasando por pruebas y validaciones en otros entornos similares al que se le ofrece al cliente, para que estas pruebas puedan ser replicadas y testeadas.

Siguiendo esta premisa, el tener diferentes versiones del proyecto que apunten a los diferentes entornos ayuda al ciclo de vida de desarrollo y mantenimiento del software. En este proyecto, por simplicidad, se usará solamente un entorno de desarrollo que será la versión final de la aplicación. Sin embargo, trabajando de esta forma y haciendo uso de la herramienta Git, se tiene la posibilidad de poder implementar una integración continua real de proyecto. Si en este proyecto participasen diferentes desarrolladores, un equipo de SDET (Software Design Engineer in Testing) y un RM (Release Manager), el proyecto a nivel de entornos debería estructurarse como se muestra en la figura 5.2.

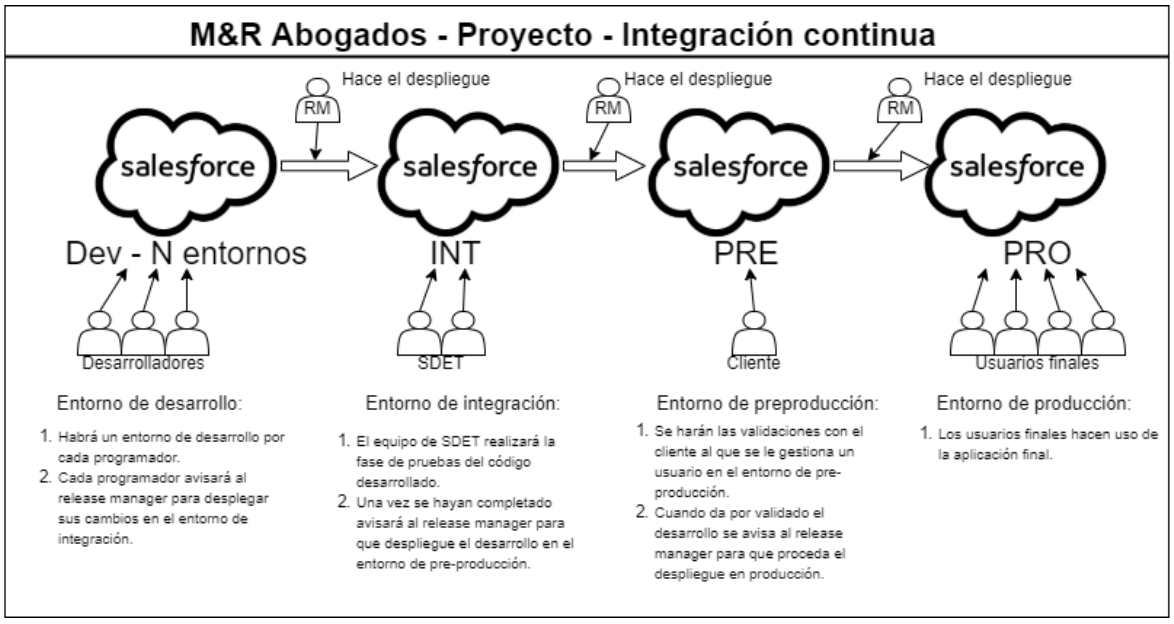


Figura 5.2 Modelo de entornos en integración continua

Llevando este modelo al control de versiones, se tendría la estructura de ramas en Git que se muestra en la figura 5.3.

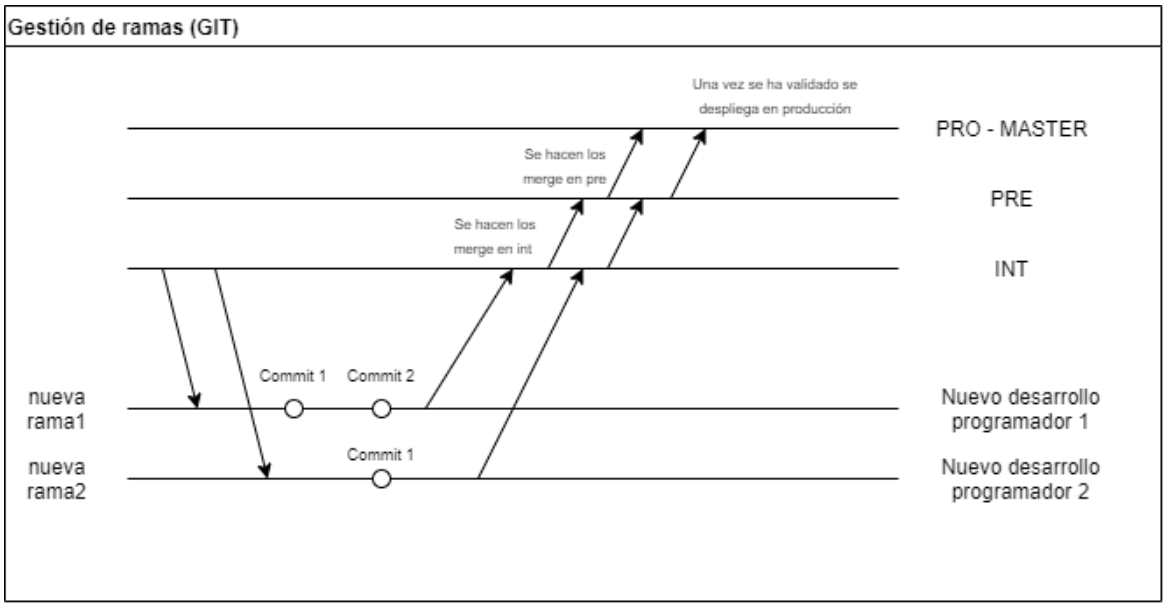


Figura 5.3 Gestión de ramas en GIT

Cada vez que hay un nuevo desarrollo, se crea una nueva rama desde INT con el nombre de la historia de usuario que se ha añadido en el Kanban. De esta forma, cada programador puede trabajar en paralelo y, una vez finalice su desarrollo, desplegar sus cambios en el entorno de integración para que sea testeado. Con esto, se mantienen los diferentes entornos alineados.

6. Pruebas

Una vez se ha completado la fase de implementación, o en paralelo a ella, es conveniente llevar a cabo las pruebas necesarias para asegurar tanto el funcionamiento como la calidad del producto que se está entregando.

En este capítulo se recogen las diferentes pruebas que se han realizado, divididas en los cuatro niveles habituales de prueba: pruebas unitarias, pruebas de integración, pruebas de sistema y de pruebas de aceptación.

6.1 Pruebas unitarias

Las pruebas unitarias, como su nombre indica, corresponden a la prueba aislada de cada unidad o módulo de la aplicación. En concreto, para cada clase Apex se ha implementado su correspondiente clase de prueba. Con esto, se asegura que cada módulo funcionará correctamente de forma independiente. Por ejemplo, si en un futuro es necesario desplegar un solo componente en otro entorno, se incluirá la clase de prueba correspondiente a la clase Apex.

Para crear una clase de prueba en Salesforce, es necesario a tener en cuenta las siguientes consideraciones:

- Para diferenciar una clase Apex (.cls) normal de una clase de test se escribirá la anotación *@isTest* encima de la declaración de la clase y de los métodos.
- Para hacer la prueba de la funcionalidad que se quiere testear, será necesario (casi siempre) una función en la cual se inicialicen los objetos mock correspondientes. Por ejemplo, si se quiere probar una funcionalidad de expedientes, en esta función se creará un expediente de prueba y se asignarán los valores a los campos correspondientes.
- Cada método de la clase de prueba hará referencia al método que se quiere testear. Los métodos de prueba no deben devolver ni recibir ningún parámetro.
- En cada método de prueba se debe incluir, al menos, una sentencia *System.assert*, que compruebe o compare los valores obtenidos con los esperados.

Tras verificar que se siguen todos estos pasos, se puede proceder a su implementación. Una vez finalizada, se procederá a la ejecución única, global o grupal de las clases de prueba. Cabe destacar que Salesforce impone la necesidad de cubrir al menos un 75% de las líneas de código Apex en cada ejecución. Si no se cubre dicho porcentaje en el entorno de desarrollo, la funcionalidad puede ejecutarse y funcionar perfectamente. Sin embargo, el problema viene cuando se quiere desplegar en un entorno productivo, donde sí que será obligatorio superar este porcentaje.

A continuación, en la figura 6.1, se muestra una ejecución global de todas las clases de prueba del sistema, mostrando la cobertura de todo el código.

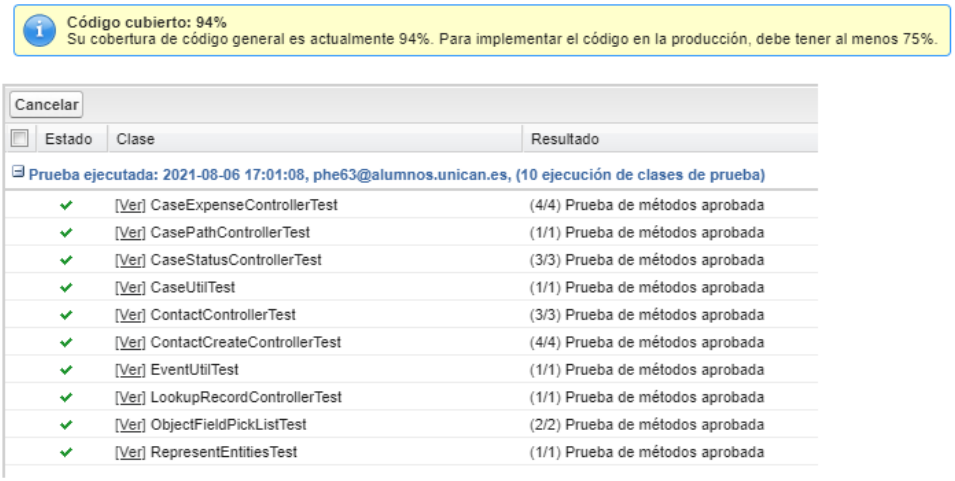
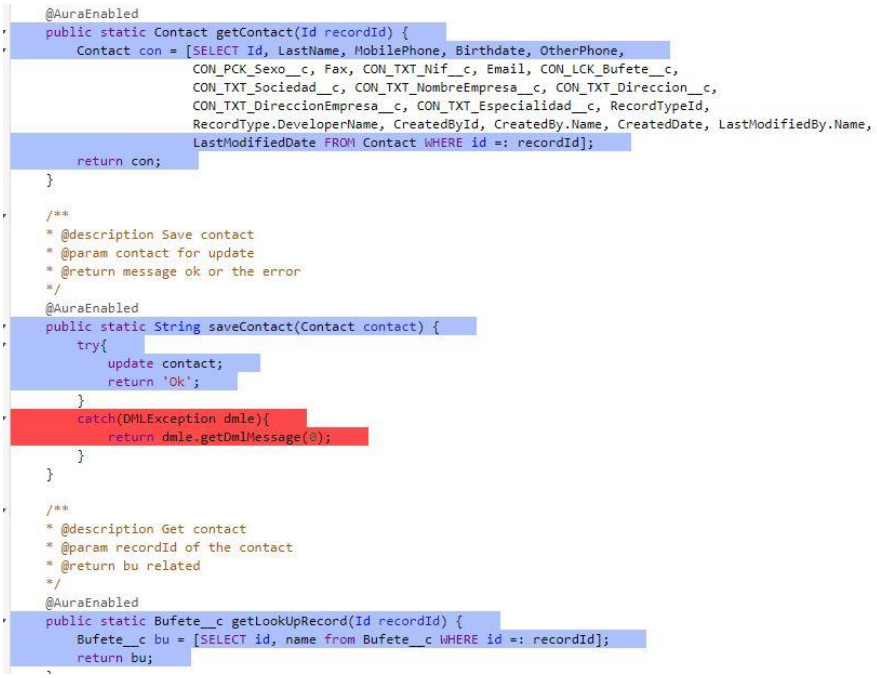


Figura 6.1 Ejecución de las clases de test

Para lanzar las clases de test de forma unitaria, se puede lanzar desde la configuración del entorno, o desde la Developer Console cuando estamos programado una clase de prueba. Después de lanzar la ejecución, pinchando en la clase que se quiere testear, se pueden ver las líneas por las que pasa al ejecutar la clase de test marcando en rojo por las que no se pasa. Se muestra un ejemplo en la siguiente figura 6.2.



```
    @AuraEnabled
    public static Contact getContact(Id recordId) {
        Contact con = [SELECT Id, LastName, MobilePhone, Birthdate, OtherPhone,
                       CON_PCK_Sexo__c, Fax, CON_TXT_Nif__c, Email, CON_LCK_Bufete__c,
                       CON_TXT_Sociedad__c, CON_TXT_NombreEmpresa__c, CON_TXT_Direccion__c,
                       CON_TXT_DireccionEmpresa__c, CON_TXT_Especialidad__c, RecordTypeId,
                       RecordType.DeveloperName, CreatedById, CreatedBy.Name, CreatedDate, LastModifiedBy.Name,
                       LastModifiedDate FROM Contact WHERE id =: recordId];
        return con;
    }

    /**
    * @description Save contact
    * @param contact for update
    * @return message ok or the error
    */
    @AuraEnabled
    public static String saveContact(Contact contact) {
        try{
            update contact;
            return 'Ok';
        }
        catch(DMLException dmle){
            return dmle.getDmlMessage(0);
        }
    }

    /**
    * @description Get contact
    * @param recordId of the contact
    * @return bu related
    */
    @AuraEnabled
    public static Bufete__c getLookUpRecord(Id recordId) {
        Bufete__c bu = [SELECT id, name from Bufete__c WHERE id =: recordId];
        return bu;
    }
```

Figura 6.2 Cobertura de una clase de test

6.2 Pruebas de integración

Las pruebas de integración verifican que las piezas del sistema software interaccionan entre sí de manera correcta, por lo que aplicadas al más alto nivel, esto es, al sistema completo, garantizan que el desarrollo funciona y hace lo que se espera de él.

En este caso las pruebas han sido realizadas de manera manual según se iba implementando cada desarrollo, ya que en base a la metodología Agile y al modelo Kanban, se debe pasar por una fase denominada “Testing” que corresponde a las pruebas de integración. Se debe de pasar por dicha fase en cada sprint con cada nuevo desarrollo. Para ello, es necesario ponerse en la piel del usuario, simular todos los casos posibles que puedan darse y volver a la fase de implementación si se encontrase cualquier tipo de error.

Al realizar los desarrollos se ha utilizado un usuario con un perfil especial llamado System Administrator, que como su propio nombre indica, se trata de un usuario de tipo administrador. Este tipo de usuario se diferencia de los demás porque tiene ciertos privilegios (acceso a la configuración, acceso a la Developer Console, registros, etc.) respecto a los usuarios normales que van a hacer uso de la aplicación. Sin embargo, para realizar las pruebas de integración se debe adoptar el rol del usuario final, por eso no es conveniente realizar las pruebas con el mismo usuario con el que se realizan los desarrollos. La plataforma de Salesforce, en entornos previos al de producción, permite acceder a la aplicación bajo la vista de otros usuarios, siempre y cuando se haga a través de un usuario con perfil System Administrator. Por ende, cada vez que se ha realizado una prueba se ha verificado la funcionalidad utilizado el/los usuario/s correspondientes.

Una vez se ha accedido a la aplicación bajo la vista del usuario seleccionado, se puede simular el funcionamiento tal y como lo vería el usuario final. Esto puede evitar ciertos problemas (de visibilidad o de algún ajuste de configuración que no se haya asignado correctamente), ya que el usuario administrador suele tener muchos más privilegios y ajustes que para él no se necesitan configurar, pero para otros usuarios sí que son necesarios. Así se asegura el correcto funcionamiento. En la figura 6.4, se muestra como ejemplo la funcionalidad de cambiar el estado de un expediente, visto con un usuario con perfil Abogado.

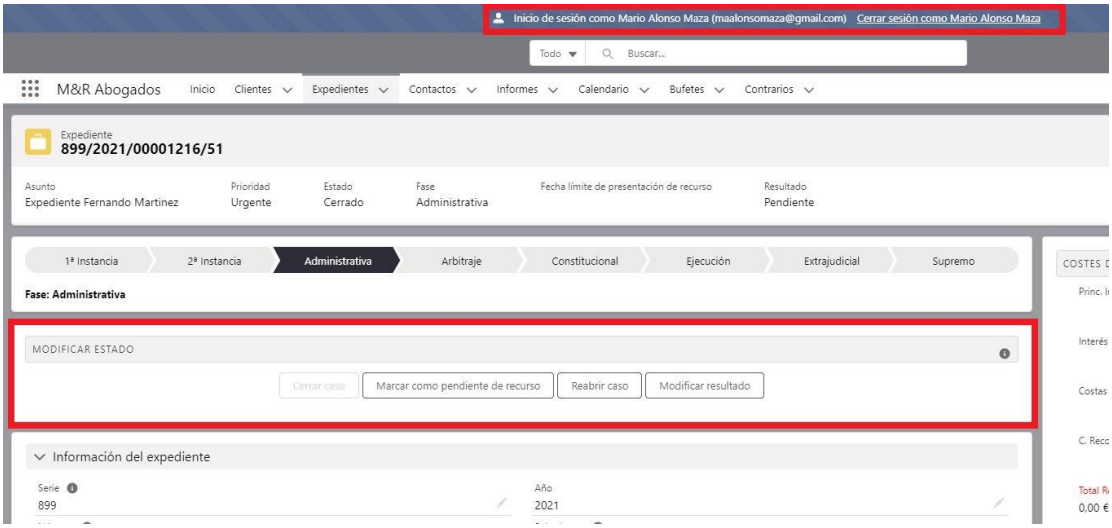


Figura 6.3 Vista de un expediente con un usuario de la aplicación

Por otro lado, para ejecutar las pruebas ha sido necesaria la inserción de datos de prueba en su versión inicial. Dataloader es la herramienta que se ha usado con este propósito, ya que permite insertar varios registros a la vez. Lo único que se ha tenido que hacer ha sido rellenar unas hojas de Excel para cada entidad, poniendo en la cabecera del archivo los campos deseados y en las filas inferiores los datos de los registros. Así, con unos pocos clics se consigue introducir de golpe todos los datos de expedientes, contactos, clientes, etc; como se puede observar en la figura 6.4.

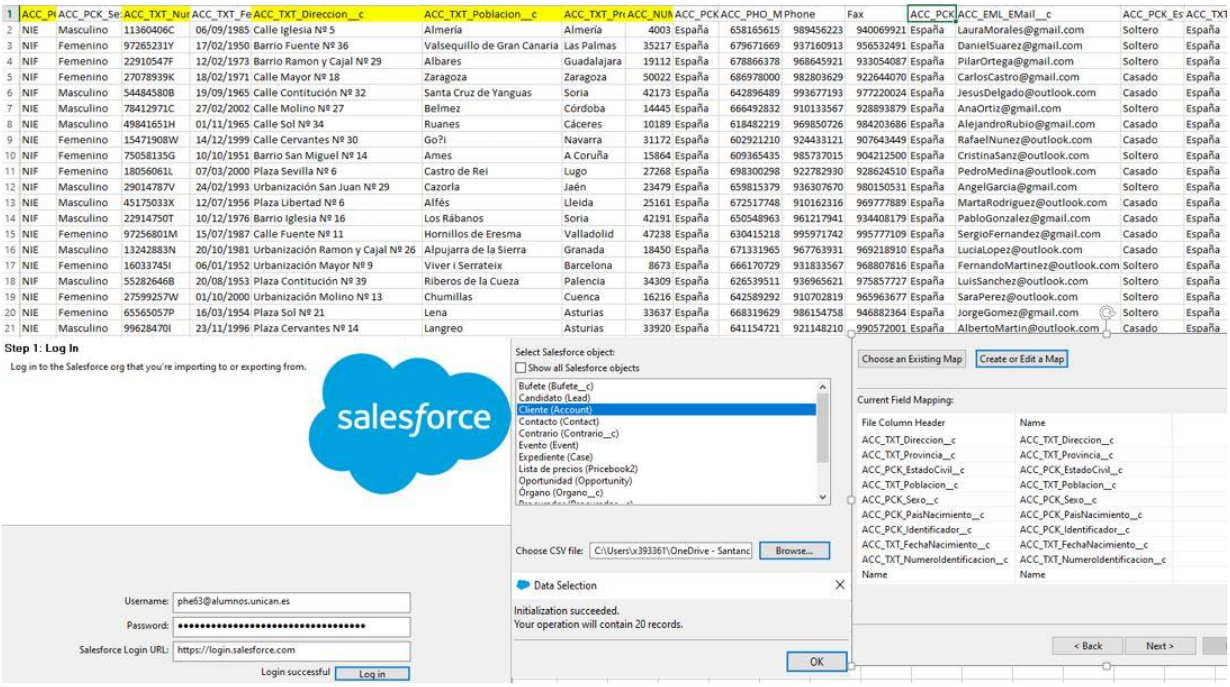


Figura 6.4 Inserción de registros mediante la herramienta Dataloader

Esta práctica no solo tiene por qué darse para hacer cargas de registros masivas, si no que puede usarse como método de regularización de datos en cualquier momento de vida de la aplicación. Por ejemplo, si se necesita actualizar el estado de cien expedientes a la vez, por el motivo que sea, sería bastante tedioso realizarlo de forma manual.

6.3 Pruebas de sistema

Las pruebas de sistema consisten en verificar que la aplicación completa cumple con todos los requisitos especificados para ella. Al haber realizado las pruebas de integración aplicando un enfoque end-to-end, todos los requisitos funcionales ya han sido probados en dicha fase y solo queda probar los requisitos no funcionales:

- En primer lugar, se ha creado un árbol jerárquico de roles dentro de la aplicación con la nomenclatura apropiada para este negocio. Esto restringirá al usuario el acceso a ciertos registros dependiendo del rol que dispongan. En la figura 6.5 se muestra el árbol jerárquico de roles de la aplicación. Por ejemplo, un usuario con rol Asociado Senior no podrá acceder a los expedientes de otro usuario de ese mismo nivel. Sin embargo, un usuario con rol superior si podrá ver los expedientes de sus subordinados.

  También se ha asignado a cada usuario un perfil determinado. Esto, a diferencia de los roles, limita al usuario qué puede hacer o que no dentro de la aplicación. Por ejemplo, se puede limitar la visibilidad de un usuario para que no tenga posibilidad de interacción con la entidad de contrato y que esta entidad sea solo utilizada por la gente de RRHH. Para verificar que todas estas premisas se cumplen, se han realizado pruebas con diferentes roles y perfiles, comprobando que, en efecto, las restricciones de acceso funcionan según lo esperado.

Figura 6.5 Jerarquía de roles de la aplicación

- En segundo lugar, Salesforce permite la configuración de seguridad de múltiples factores para el acceso a la aplicación. Este método de seguridad se activa cuando iniciemos sesión en un nuevo navegador o dispositivo, y a parte de introducir la clave, pedirá al usuario un código que llega al correo electrónico o al teléfono personal.

- En tercer lugar, la aplicación ha sido probada tanto desde el navegador como desde la aplicación móvil/tablet de Salesforce, verificando que se cumple con el requisito de portabilidad correspondiente.

- Por último, se ha comprobado que el usuario puede cambiar el idioma de la aplicación en todo momento, y que en cada caso, todo el texto aparece en el idioma deseado.

6.4 Pruebas de aceptación

Una vez se han completado las pruebas de sistema, el siguiente paso es llevar a cabo las pruebas de aceptación. En estas pruebas el usuario final verifica que el comportamiento de la aplicación es el esperado y a su vez, se comprueba su grado de satisfacción con el producto entregado.

En este caso, se ha proporcionado un usuario de la aplicación a un usuario final con el que ha podido realizar las pruebas y comprobar que la funcionalidad de la aplicación es la que esperaba. Siguiendo el método Agile, estas pruebas se han realizado tanto en cada uno de los desarrollos de forma individual, como con la aplicación final una vez terminada la primera versión del proyecto. El resultado de las pruebas ha sido satisfactorio aunque también han derivado en algunas propuestas de mejora.

7. Conclusiones y trabajos futuros

Tras la finalización del proyecto ha de hacerse un análisis del trabajo realizado para sacar las diferentes conclusiones de este. Estas conclusiones serán utilizadas para cambios o mejoras a implementar en trabajos futuros.

Es por ello que, en este capítulo, primero se expondrán las conclusiones, dificultades y mejoras a llevar a cabo y, en segundo lugar, se expondrán los trabajos a realizar a futuro.

7.1 Conclusiones

En resumen, el objetivo principal de este proyecto ha sido la creación de una aplicación en la nube, utilizando el CRM Salesforce, que tuviese la función de gestión de datos sobre un negocio. En este caso el negocio elegido ha sido un bufete de abogados, pero podría haber sido para cualquier otro tipo de negocio en el cual haya conversaciones, negociaciones o cualquier tipo de contacto continuo con el cliente.

Se puede concluir que todos los requisitos recogidos en los requisitos funcionales y no funcionales han sido implementados y, seguidamente, validados por el usuario final. Lo que ha derivado en la primera versión de la aplicación y, a su vez, en el trabajo final de este trabajo de fin de grado. Esto no quiere decir que la aplicación esté finalizada, sino que hay una primera versión funcional y operacional con la que el usuario final puede desempeñar su trabajo.

Alguna de las dificultades encontradas han sido, por ejemplo, a la hora de hacer la toma de requisitos del caso de modificar el estado de un expediente, pues el usuario se encontró con una funcionalidad que no era la esperada. Sin embargo, los cambios que se tuvieron que hacer para arreglarlo no fueron muy complicados. Esto sirvió para aprender a obtener información más útil en la toma de requisitos. Por otro lado, también se han encontrado algunas dificultades típicas a nivel de programación, con fallos desconocidos que han requerido indagar en internet para encontrar la respuesta.

Las mejoras que se han planteado han sido, por un lado, modificar el componente de gestión de costes del expediente para añadir más información, y por otro, añadir más campos a la entidad de bufetes que ahora mismo guarda muy poca información.

7.2 Trabajos futuros

La inmensa mayoría de los trabajos futuros aparecerán cuando el usuario final utilice la aplicación en su día a día y vea necesidades adicionales a las que se plantearon en la versión inicial. Sin embargo, ya han aparecido algunos nuevos requisitos:

- Desarrollo de un nuevo módulo de la aplicación en el que se realicen las gestiones de RRHH de la empresa para poder tener una visión de la gestión de contratos, estadísticas de rendimiento de los abogados del bufete, ganancias y pérdidas, contratos con proveedores, etc.
- Añadir la sincronización del correo con Outlook para que los eventos que se creen en Salesforce se sincronicen con el calendario del correo electrónico y viceversa.

- Añadir una nueva entidad donde almacenar datos sobre los diferentes tipos de interacciones con el cliente, terceras personas, contactos, etc; que esté relacionada con los las demás entidades.

BIBLIOGRAFÍA

[1] Salesforce, Guía para principiantes sobre los sistemas de CRM, https://www.salesforce.com/es/learning-centre/crm/crm-systems/ [Último acceso: 02-06-2021]

[2] Salesforce, https://www.salesforce.com [Último acceso: 13-06-2021]

[3] Lauren Horwitz, Aura framework, https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/intro_framework.htm [Último acceso: 13-06-2021]

[4] Salesforce, Apex Code Overview, https://help.salesforce.com/s/articleView?id=sf.code_about.htm&type=5 [Último acceso: 13-06-2021]

[5] Salesforce, Ejecutar consultas SOQL y SOSL, https://trailhead.salesforce.com/es-MX/content/learn/modules/developer_console/developer_console_queries [Último acceso: 24-07-2021]

[6] Developer Salesforce, Salesforce Extensions for Visual Studio Code, https://developer.salesforce.com/tools/vscode [Último acceso: 28-07-2021]

[7] Ian Sommerville (2009), Software Engineering 9th Edition, Addison-Wesley.

[8] Nevenka Kirovska y Saso Koceski, USAGE OF KANBAN METHODOLOGY AT SOFTWARE DEVELOPMENT TEAMS, http://www.aebjournal.org/articles/0303/030302.pdf [Último acceso: 28-07-2021]

[9] Vardhan NS, Get Started With Salesforce Programming, https://medium.com/edureka/salesforce-developer-1051ba8ce733 [Último acceso: 03-08-2021]

[10] Developer Salesforce, Source Commands, https://developer.salesforce.com/docs/atlas.en-us.sfdx_cli_reference.meta/sfdx_cli_reference/cli_reference_force_source.htm [Último acceso: 18-08-2021]